

IDEA:

1) Al igual que en los ejercicios que venimos haciendo, intentamos hacer la parte de "Divide" partiendo el arreglo en dos. Para no tener el costo de copiar el arreglo, usamos una función auxiliar MaxAlfajoresAux(cajas,i,j) donde i y j son naturales que funcionan como delimitadores. Luego tenemos:

```
MaxAlfajores(cajas){
    return MaxAlfajoresAux(cajas, 0, cajas.size())
}
```

```
MaxAlfajoresAux(cajas, i, j){
    if (j-i=1)
    then
        return CASO BASE
    else
        mitad = (i+j)/2

        izq = MaxAlfajoresAux(cajas, i, mitad)
        der = MaxAlfajoresAux(cajas, mitad, j)

        CONQUER

        return RES
}
```

2) Ahora con el esquema, una primera idea sería que devolvamos la mayor cantidad de alfajores que se pueden conseguir en el juego respetando las restricciones. Esto nos deja el caso base de la siguiente manera:

```
MaxAlfajoresAux(cajas, i, j){
    if (j-i=1)
    then
        return cajas[i]
    else
        mitad = (i+j)/2

        izq = MaxAlfajoresAux(cajas, i, mitad)
        der = MaxAlfajoresAux(cajas, mitad, j)

        CONQUER

        return RES
}
```

3) Ahora bien, hay que pensar cómo obtenemos la mayor cantidad de alfajores que se pueden conseguir en el juego respetando las restricciones, teniendo dichas cantidades para la mitad izquierda y para la mitad derecha del arreglo. Acá es donde entra en juego el ejercicio que vimos en la clase sincrónica de D&C. Si hicieramos izq+der, podríamos estar considerando casos donde la máxima cantidad de alfajores que podemos conseguir del lado izquierdo cosnsidera los alfajores de la caja en su extremo derecho, y la máxima cantidad de alfajores que podemos conseguir del lado derecho cosnsidera los alfajores de la caja en su extremo izquierdo, algo que en realidad es inválido puesto que al seleccionar una caja, las aldeañas se vacían. Luego una opción es:

```
MaxAlfajoresAux(cajas, i, j){
    if (j-i=1)
    then
        return cajas[i]
    else if (j-i<1) then
        return 0
    else
        mitad = (i+j)/2
```

```

    izq = MaxAlfajoresAux(cajas, i, mitad)
    der = MaxAlfajoresAux(cajas, mitad, j)
    izqSinDer = MaxAlfajoresAux(cajas, i, mitad-1)
    derSinIzq = MaxAlfajoresAux(cajas, mitad+1, j)

    res = max(izqSinDer+der, derSinIzq+izq)

    return res
}

```

4) Sin embargo, nos queda una recurrencia de la pinta $4T(n/2)+O(1)$ y por el Teorema Maestro nos da una complejidad de $\Theta(n^2)$ y no cumple lo pedido

Nuevamente, usamos una "estrategia" vista en clase, de guardarnos los resultados que necesitamos en la recurrencia en una tupla, en este caso <mayor cantidad de alfajores que se pueden conseguir en el juego respetando las restricciones, idem pero sin la caja del comienzo, idem pero sin la caja del final>:

```

MaxAlfajoresAux(cajas, i, j){
    if (j-i=1)
    then
        return <cajas[i],0,0>
    else
        mitad = (i+j)/2

        izq = MaxAlfajoresAux(cajas, i, mitad)
        der = MaxAlfajoresAux(cajas, mitad, j)

        completo_izq = pi1(izq)
        sin_primerio_izq = pi2(izq)
        sin_ultimo_izq = pi3(izq)

        completo_der = pi1(der)
        sin_primerio_der = pi2(der)
        sin_ultimo_der = pi3(der)

        completo_opcion_1 = completo_izq + sin_primerio_der
        completo_opcion_2 = sin_ultimo_izq + completo_der
        completo = max(completo_opcion_1, completo_opcion_2)

        sin_primerio = ??
        sin_ultimo = ??

        return res
}

```

5) Ahora lo que tenemos que pensar es cómo obtener los valores que devolvemos en las tuplas a partir de los que tenemos (similar al ejercicio de triplicados donde ahora no solo teníamos que calcular triples, sino también dobles y simples). Resolver nuestro problema sin considerar la primera caja, considera los casos donde: no usamos la primera caja de la izquierda (pudiendo o no usar la última), por lo que lo combinamos con lo que tenemos de no usar la primera caja de la derecha; o bien no usamos la primera NI LA ULTIMA caja de la izquierda, y lo combinamos con lo que tenemos de poder usar todas las cajas de la derecha. Lo simétrico sucede al querer calcular el máximo sin considerar la última caja. Sin embargo, esto nos pide entonces algo que no teníamos: el máximo sin la primer NI ultima caja, por lo que tenemos que devolver en la tupla que calculamos. Y nuevamente, tenemos que ver cómo obtenerlo a partir de los resultados de la izquierda y la derecha (sale con las mismas ideas). Finalmente nos queda:

```

MaxAlfajores(cajas){
    return MaxAlfajoresAux(cajas, 0, cajas.size())
}

```

```

MaxAlfajoresAux(cajas, i, j){
  if (j-i=1)
  then
    return <cajas[i],0,0,0> //O(1)
  else
    mitad = (i+j)/2

    izq = MaxAlfajoresAux(cajas, i, mitad) //T(n/2)
    der = MaxAlfajoresAux(cajas, mitad, j) //T(n/2)

    //Las siguientes son todas operaciones elementales por lo que son O(1)

    completo_izq = pi1(izq)
    sin_primerio_izq = pi2(izq)
    sin_ultimo_izq = pi3(izq)
    sin_p_u_izq = pi4(izq)

    completo_der = pi1(der)
    sin_primerio_der = pi2(der)
    sin_ultimo_der = pi3(der)
    sin_p_u_der = pi4(der)

    completo_opcion_1 = completo_izq + sin_primerio_der
    completo_opcion_2 = sin_ultimo_izq + completo_der
    completo = max(completo_opcion_1, completo_opcion_2)

    sin_primerio_opcion_1 = sin_primerio_izq + sin_primerio_der
    sin_primerio_opcion_2 = sin_p_u_izq + completo_der
    sin_primerio = max(sin_primerio_opcion_1 + sin_primerio_opcion_2)

    sin_ultimo_opcion_1 = completo_izq + sin_p_u_der
    sin_ultimo_opcion_2 = sin_ultimo_izq + sin_ultimo_der
    sin_ultimo = max(sin_ultimo_opcion_1, sin_ultimo_opcion_2)

    sin_p_u_opcion_1 = sin_primerio_izq + sin_p_u_der
    sin_p_u_opcion_2 = sin_p_u_izq + sin_ultimo_der
    sin_p_u = max(sin_p_u_opcion_1, sin_p_u_opcion_2)

    return (completo, sin_primerio, sin_ultimo, sin_p_u)
}

```

6) Luego nos queda una recurrencia de la forma $2T(n/2)+O(1)$ por lo que por el Teorema Maestro nos queda complejidad $\Theta(n)$, en particular $O(n)$.

OBS: En esta solución pusimos el foco en el desarrollo de la idea del algoritmo para que se entienda bien. Lo que esperabamos en el parcial, ademas del algoritmo, era que dieran la idea detras del mismo y justificaran detalladamente su funcionamiento y correctitud. Lo mismo para el caso de la complejidad.